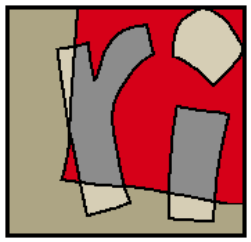
Exploration aléatoire de modèles

finis et non probabilistes

Johan Oudinet
<oudinet@lri.fr>

Laboratoire de Recherche en Informatique (LRI)
http://www.lri.fr
Université de Paris-Sud & CNRS

25 mars 2010



Plan

Test et model-checking de grands modèles

Objectif :

- explorer de grands modèles (représentés par des automates finis),
- critère de couverture : tous les chemins (d'une longueur bornée).

Problème

Un nombre exponentiel de chemins à couvrir sur un grand modèle.
$\Rightarrow$ exploration aléatoire des chemins

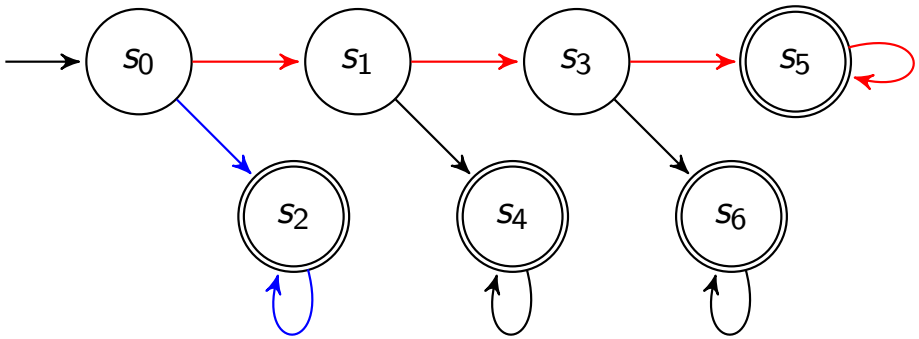
Les marches aléatoires (isotropes)

- Une marche aléatoire isotrope dans l'espace d'état du modèle (un graphe de contrôle, etc.) est :
  une séquence d'états $s_0, s_1, \ldots, s_n$ telle que s_i est choisi uniformément parmi les successeurs de l'état s_{i-1}

- Très simple à implémenter et ne demande qu'une connaissance locale du graphe.

- Beaucoup d'applications en Test (protocoles), Simulation et Model-checking (travaux récents).

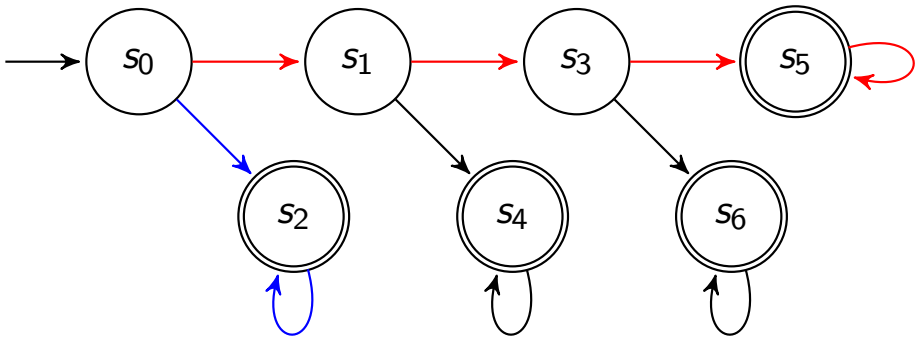
Inconvénient des marches aléatoires isotropes



$$Pr(\{s_0, s_2, s_2, s_2\}) = 0.5$$
$$Pr(\{s_0, s_1, s_3, s_5\}) = 0.125$$

- La couverture obtenue est dépendante de la topologie du graphe

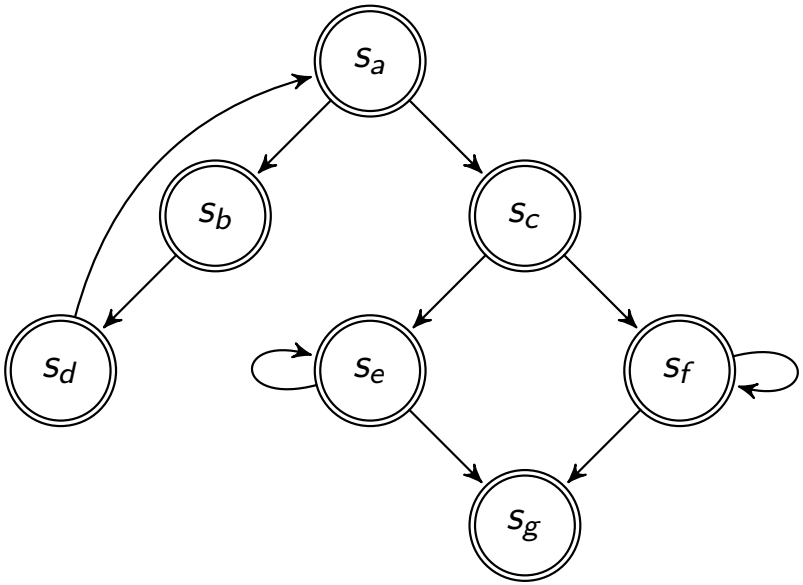
Inconvénient des marches aléatoires isotropes



$$Pr(\{s_0, s_2, s_2, s_2\}) = 0.5$$
$$Pr(\{s_0, s_1, s_3, s_5\}) = 0.125$$

- La couverture obtenue est dépendante de la topologie du graphe
- Comment équilibrer la probabilité de chaque chemin pour maximiser leur couverture ?
  $\Rightarrow$ tirage uniforme de chemins

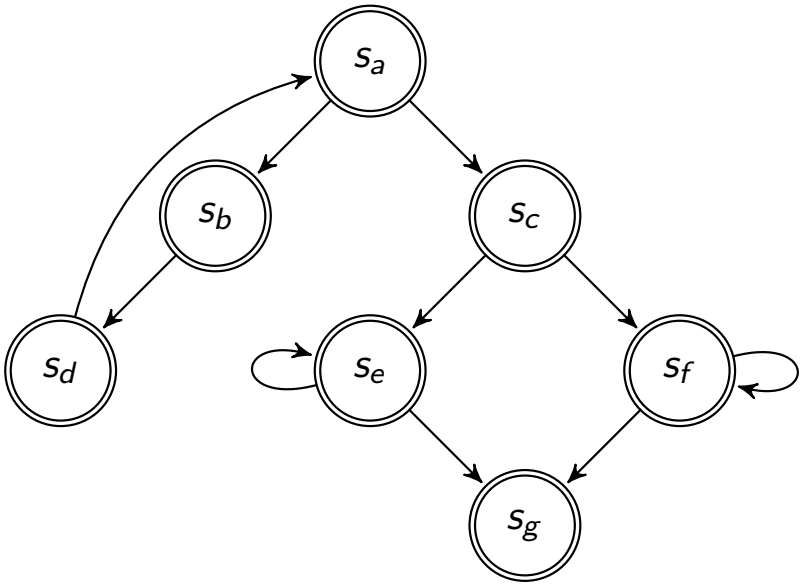
Méthode récursive [Wilf'77, Flajolet et al.'94]



$$\begin{cases} l_s(0) = 1 \\ l_s(i) = \sum_{s \to s'} l_{s'}(i-1) \quad \forall i > 0 \end{cases}$$

$$Pr(s' \mid s, i) = \frac{l_{s'}(i-1)}{l_s(i)}$$

Méthode récursive [Wilf'77, Flajolet et al.'94]

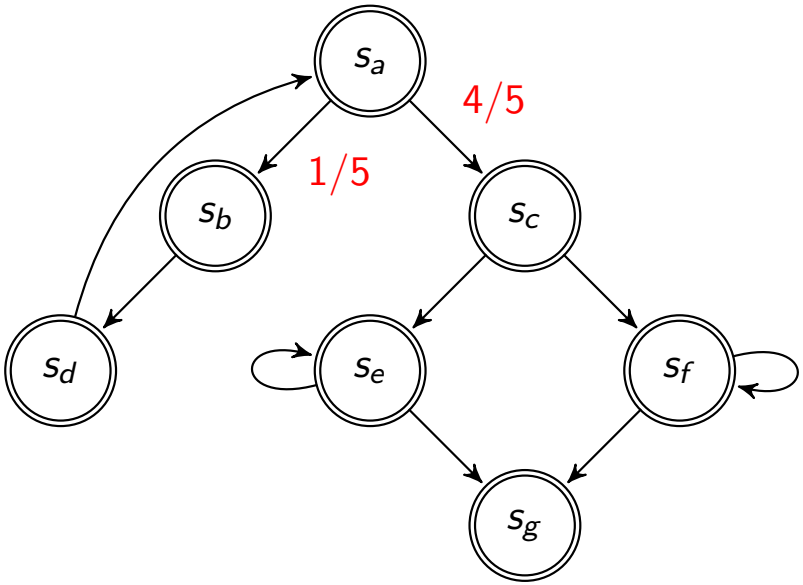


$$\begin{cases} l_s(0) = 1 \\ l_s(i) = \sum_{s \to s'} l_{s'}(i-1) \quad \forall i > 0 \end{cases}$$

$$Pr(s' \mid s, i) = \frac{l_{s'}(i-1)}{l_s(i)}$$

#	s_a	s_b	s_c	s_d	s_e	s_f	s_g
0	1	1	1	1	1	1	1
1	2	1	2	1	2	2	0
2	3	1	4	2	2	2	0
3	5	2	4	3	2	2	0

Méthode récursive [Wilf'77, Flajolet et al.'94]

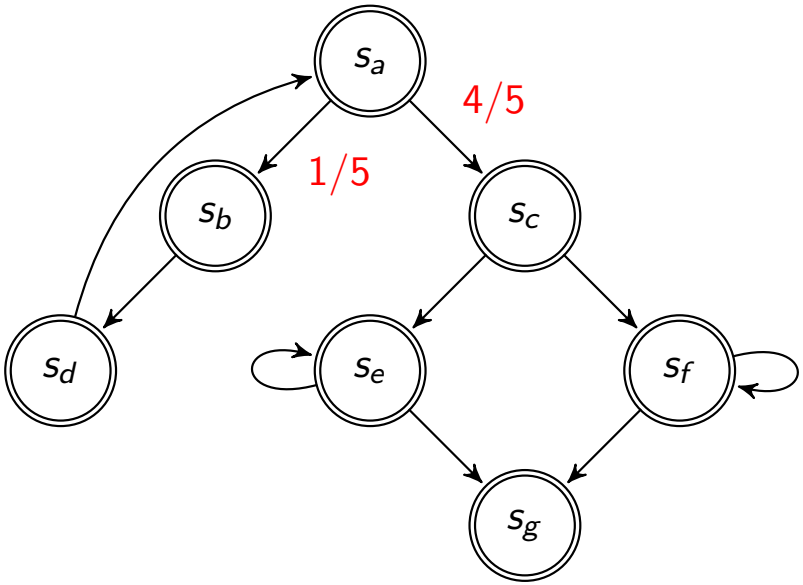


$$\begin{cases} l_s(0) = 1 \\ l_s(i) = \sum_{s \to s'} l_{s'}(i-1) \quad \forall i > 0 \end{cases}$$

$$Pr(s' \mid s, i) = \frac{l_{s'}(i-1)}{l_s(i)}$$

#	s_a	s_b	s_c	s_d	s_e	s_f	s_g
0	1	1	1	1	1	1	1
1	2	1	2	1	2	2	0
2	3	1	4	2	2	2	0
3	5	2	4	3	2	2	0

Méthode récursive [Wilf'77, Flajolet et al.'94]



$$\begin{cases} l_s(0) = 1 \\ l_s(i) = \sum_{s \to s'} l_{s'}(i-1) \quad \forall i > 0 \end{cases}$$

$$Pr(s' \mid s, i) = \frac{l_{s'}(i-1)}{l_s(i)}$$

#	s_a	s_b	s_c	s_d	s_e	s_f	s_g
0	1	1	1	1	1	1	1
1	2	1	2	1	2	2	0
2	3	1	4	2	2	2	0
3	5	2	4	3	2	2	0

- Construire la table demande $\Theta(n \times d \times q)$ opérations.
- Ensuite, tirer un chemin demande $\Theta(d \times n)$ opérations arithmétiques.

Limites de la méthode récursive

Nombre de chemins d'une longueur n

Il est généralement exponentiel en n. Chaque nombre de la table de comptage occupe donc $\mathcal{O}(n)$ bits. Et chaque opération arithmétique est en $\mathcal{O}(n)$.

$\mathcal{O}(n^2 \times q)$ bits nécessaires pour stocker la table de comptage :

- si q est très grand, tirage modulaire [Gaudel, Denise, Gouraud, Lassaigne, Oudinet et Peyronnet'08]
- si n est grand, variantes pour économiser de la mémoire.
  - sans garder la table de comptage en mémoire [Goldwurm'94]
  - calculs avec une arithmétique flottante [Denise et al.'99]

Variante sans table de comptage

- La table de comptage est construite de la ligne 0 à la ligne n avec la récurrence suivante :

$$\begin{cases} l_s(0) = 1 \\ l_s(i) = \sum_{s \to s'} l_{s'}(i-1) \quad \forall i > 0 \end{cases}$$

- Le tirage d'un chemin parcourt cette table des lignes n à 0
- Si on avait la récurrence suivante :

$$l_s(i) = \sum_{t \in \mathcal{S}} F(s, t, l_t(i+1))$$

  on pourrait éviter de conserver toute la table en mémoire en recalculant les lignes au fur et à mesure.
- Une génération plus lente mais plus économe en mémoire.

Solution des récurrences inverses (pour langages réguliers)

Solution en algèbre linéaire

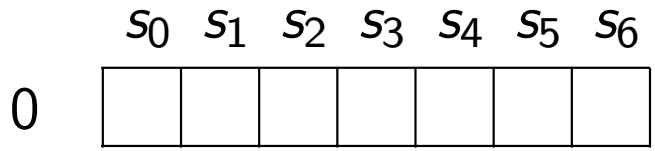
Soit $A \in \mathbb{N}^{q \times q}$ la matrice d'adjacence, on peut toujours trouver un nombre n_0 et une matrice $B \in \mathbb{Q}^{q \times q}$ tels que :

$$\forall i \geq n_0, \quad B.A^{i+1} = A^i$$

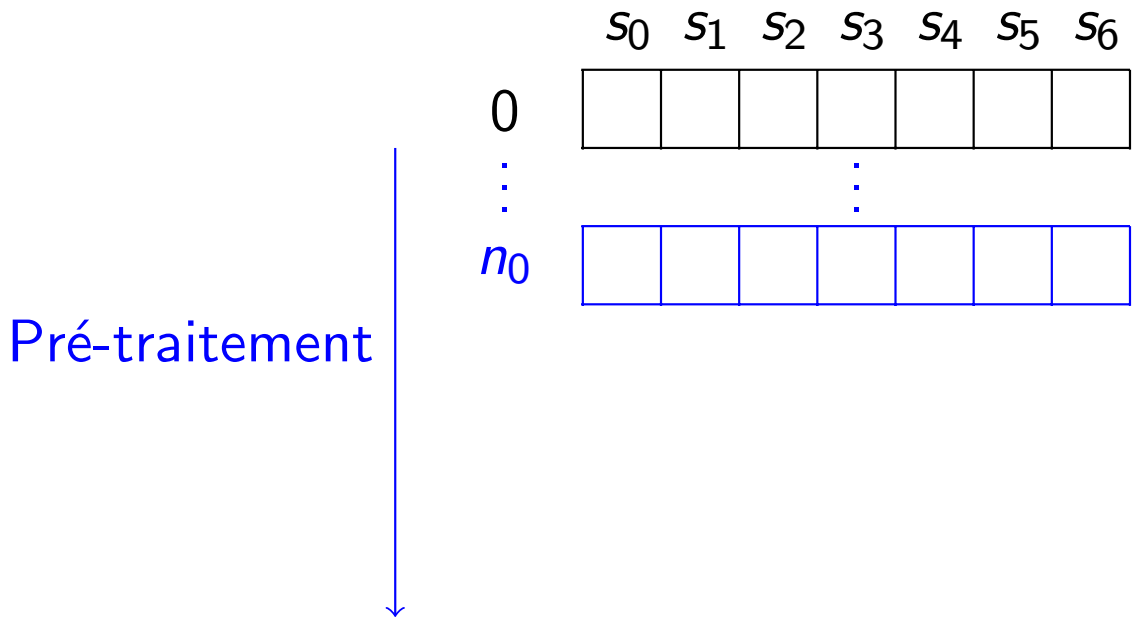
Finalement,

$$l_s(i) = \sum_{t \in S} r_{st} \times l_t(i+1) \quad \forall i > n_0$$

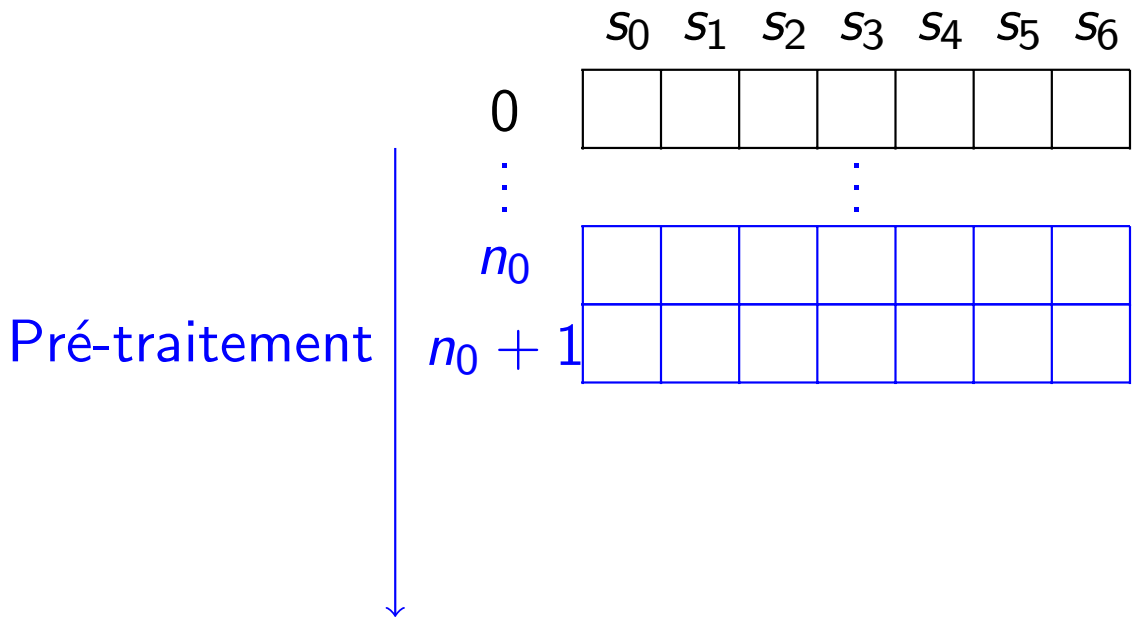
Algorithme de tirage uniforme sans table



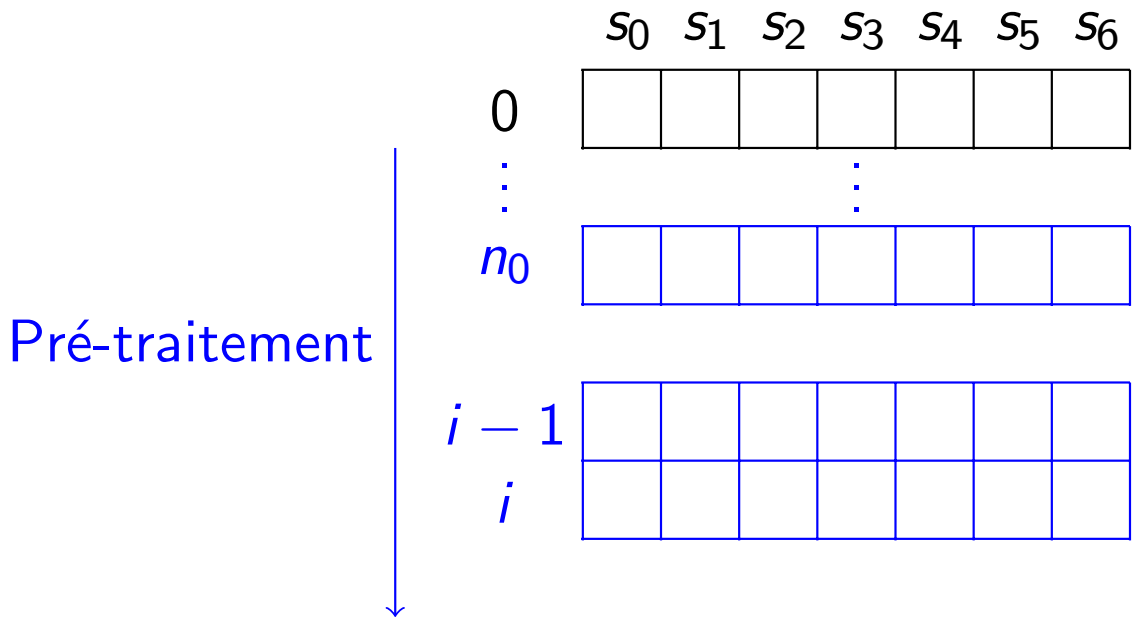
Algorithme de tirage uniforme sans table



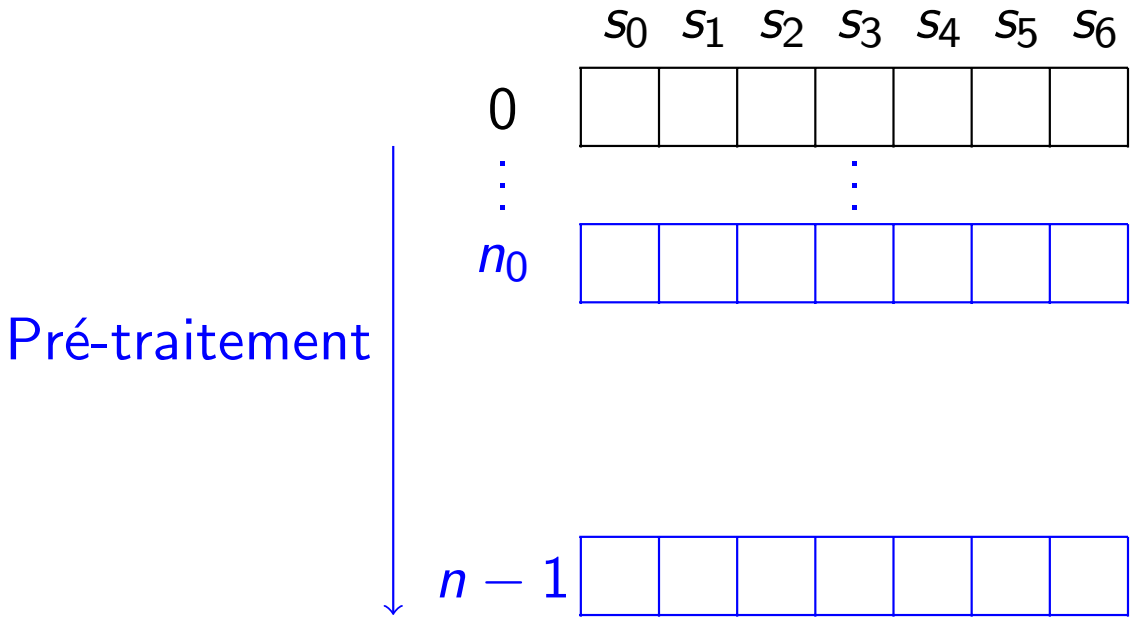
Algorithme de tirage uniforme sans table



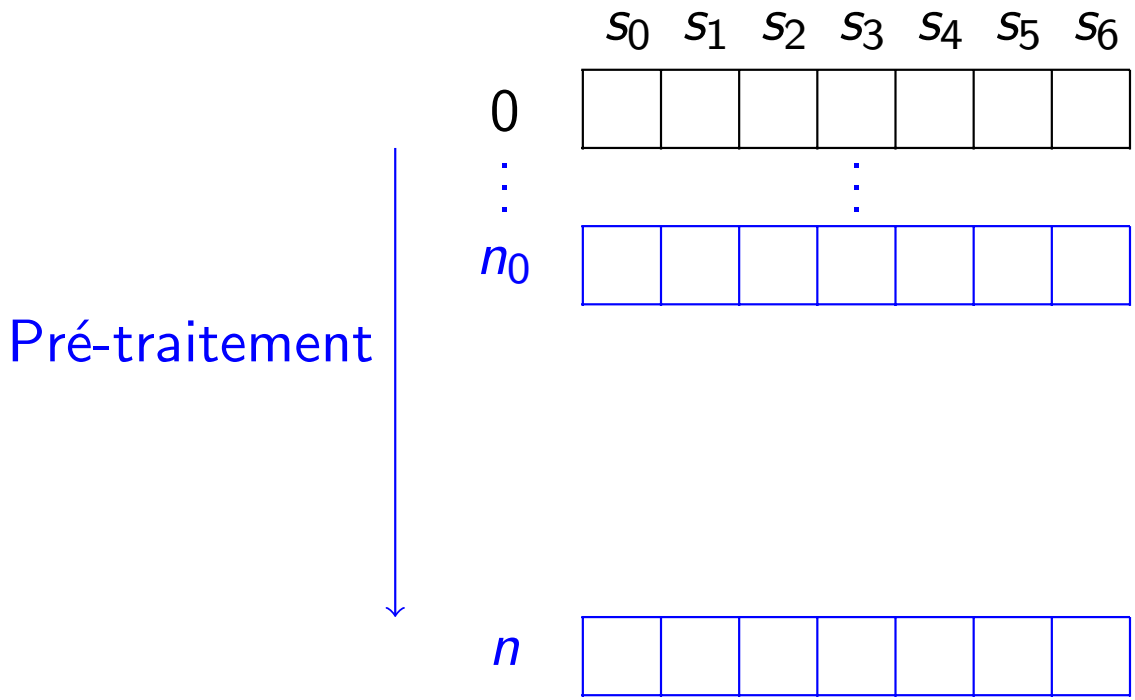
Algorithme de tirage uniforme sans table



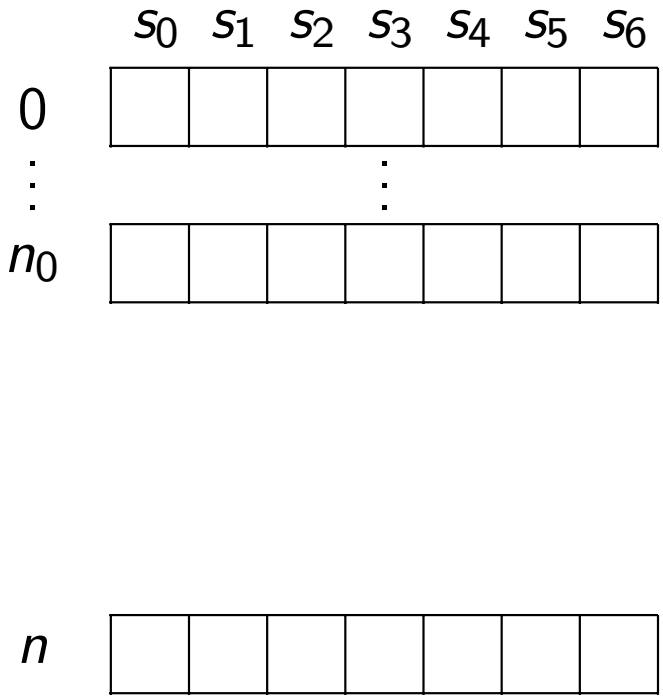
Algorithme de tirage uniforme sans table



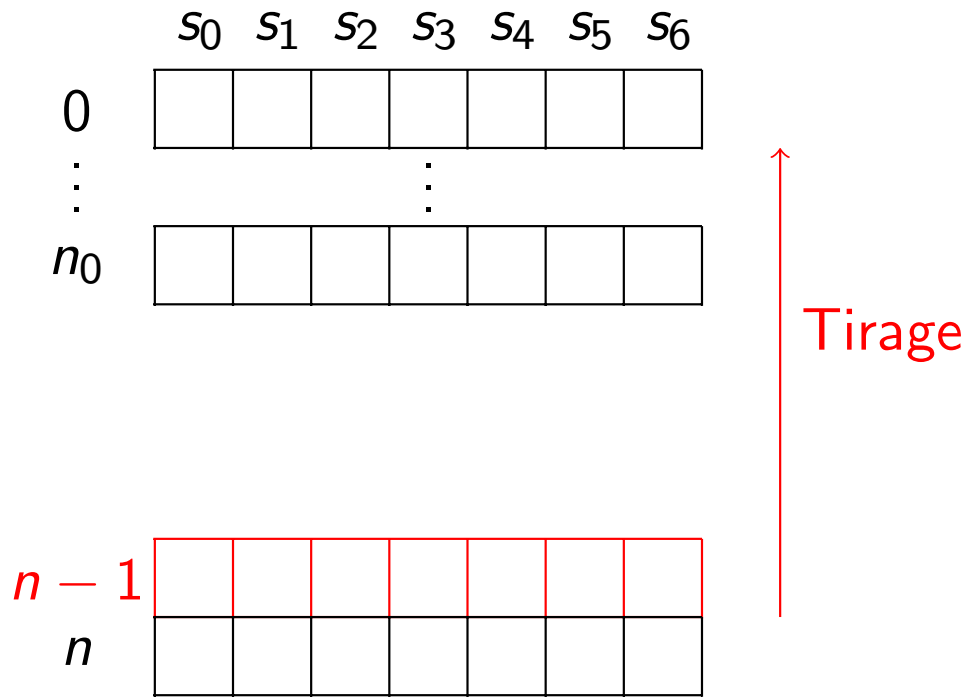
Algorithme de tirage uniforme sans table



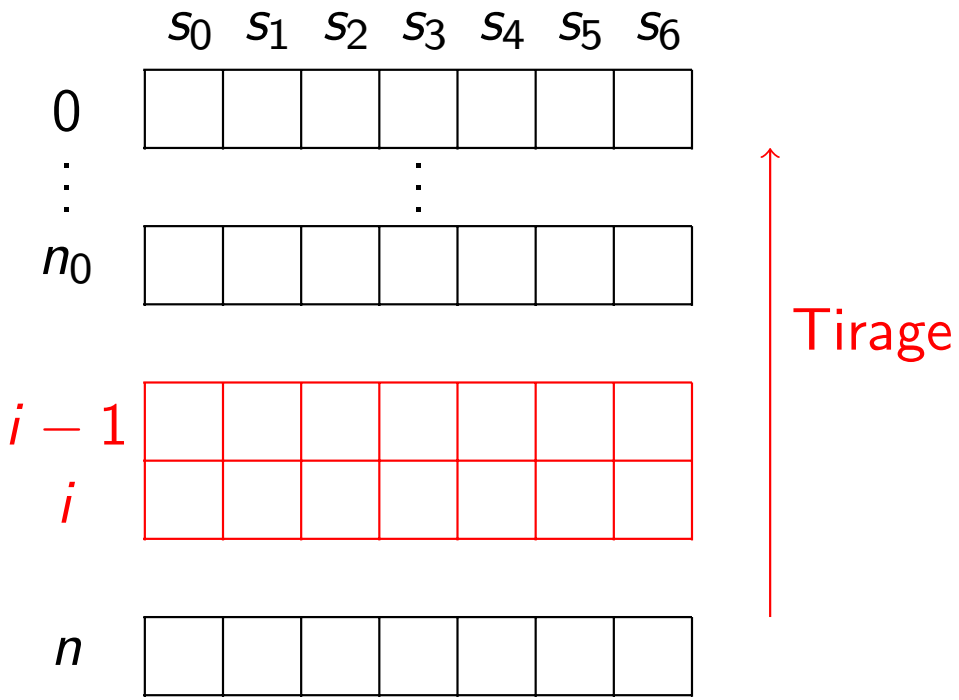
Algorithme de tirage uniforme sans table



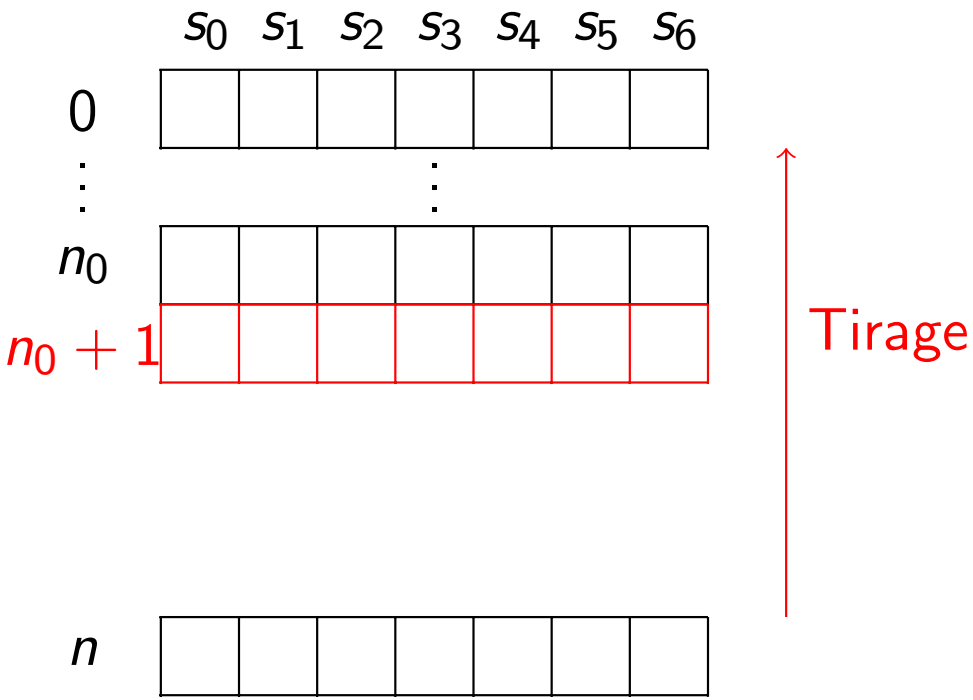
Algorithme de tirage uniforme sans table



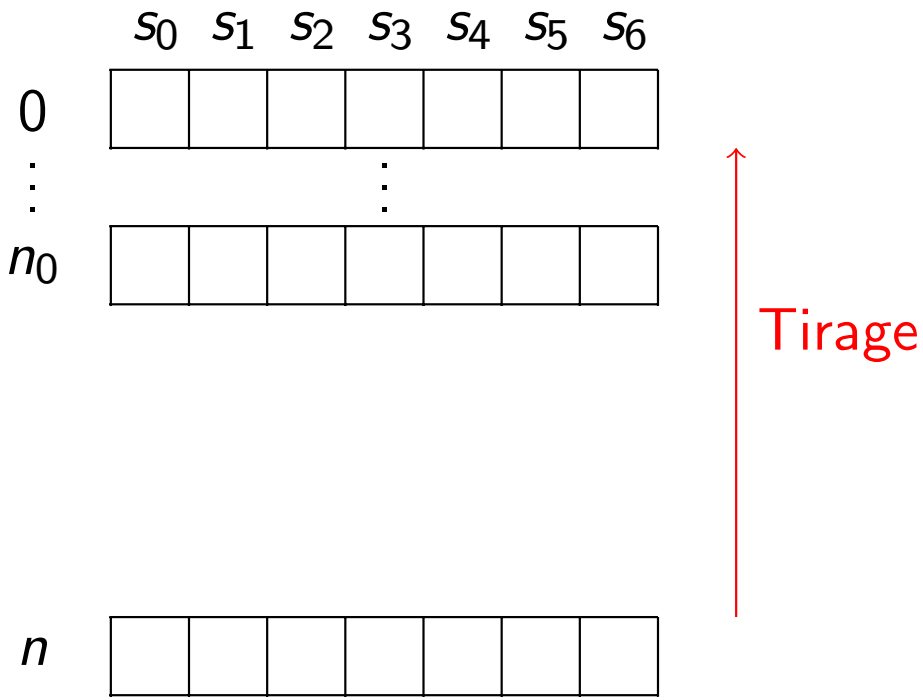
Algorithme de tirage uniforme sans table



Algorithme de tirage uniforme sans table

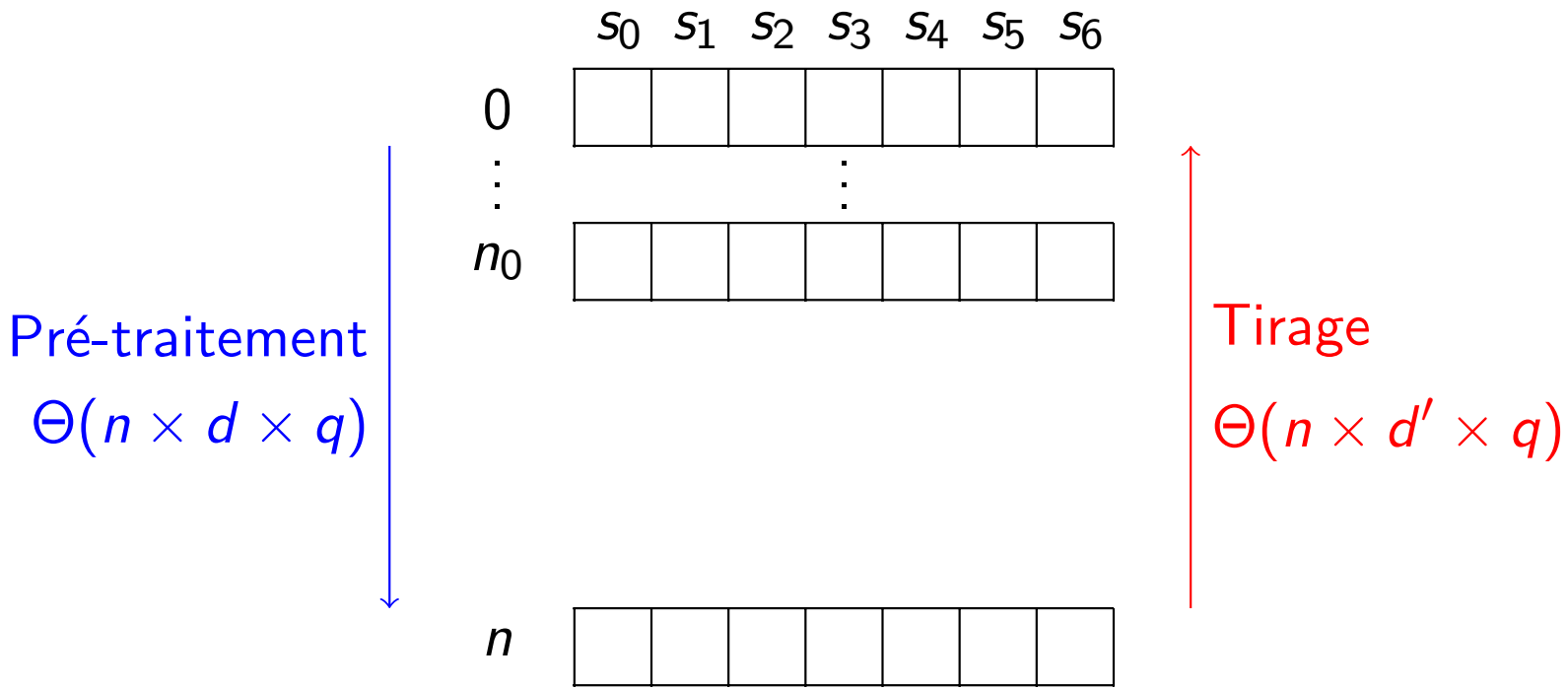


Algorithme de tirage uniforme sans table



Algorithme de tirage uniforme sans table

Mémoire : $\Theta((n_0 + 4) \times q)$ nombres



Variante avec une arithmétique flottante

- Pour gagner de la place et du temps : utiliser une arithmétique flottante à la place d'une arithmétique exacte.
- Taille de la mantisse fixe (b) et l'exposant en $\mathcal{O}(\log n)$.
- Denise et Zimmermann, 99 : borne sur l'erreur en $\mathcal{O}(n \times d \times 2^{-b})$ pour la variante avec table $\Rightarrow$ numériquement stable.
- Essais avec la variante sans table ☹ (nombres négatifs et de norme supérieure à 1)

Instabilité numérique des récurrences inverses

Les relations de récurrences inverses sont de la forme :

$$l_s(i) = \sum_{s'} r_{s,s'} \times l'_s(i+1) \quad \text{avec } r_{s,s'} \in \mathbb{Q}$$

Propagation géométrique de l'erreur :

$$\varepsilon^s_{n-1} = \sum_t \varepsilon^t_n \times |r_{st}|$$

$$\varepsilon_n = \max_s \ \varepsilon^s_n$$

$$\varepsilon_{n-1} \leq \max_s \ \sum_t \varepsilon_n \times |r_{st}|$$

$$\varepsilon_{n-1} \leq \left[\max_s \ \sum_t |r_{st}| \right] \varepsilon_n$$

Complexités binaires

Soit :

- n la longueur des chemins ;
- q le nombre d'états du système.

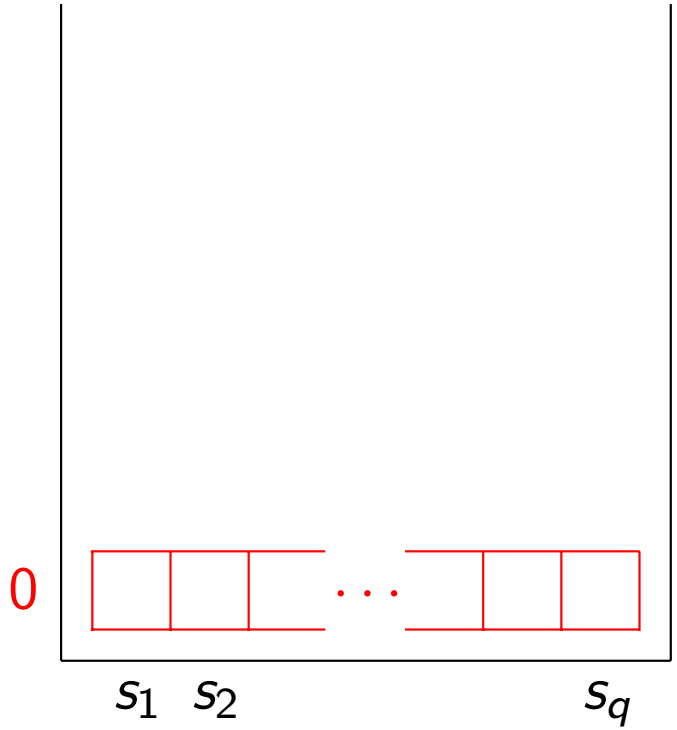
Les complexités binaires pour générer un chemin de longueur n sont :

	Calcul exact		Calcul flottant	
	Temps	Espace	Temps	Espace
avec table	$\mathcal{O}(n^2 \times d)$	$\mathcal{O}(n^2 \times q)$	$\mathcal{O}(n \log n \times d)$	$\mathcal{O}(n \log n \times q)$
sans table	$\mathcal{O}(n^2 \times d' \times q)$	$\mathcal{O}(n \times q)$	$\mathcal{O}(n \log n \times d' \times q)$	$\mathcal{O}(\log n \times q)$

Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

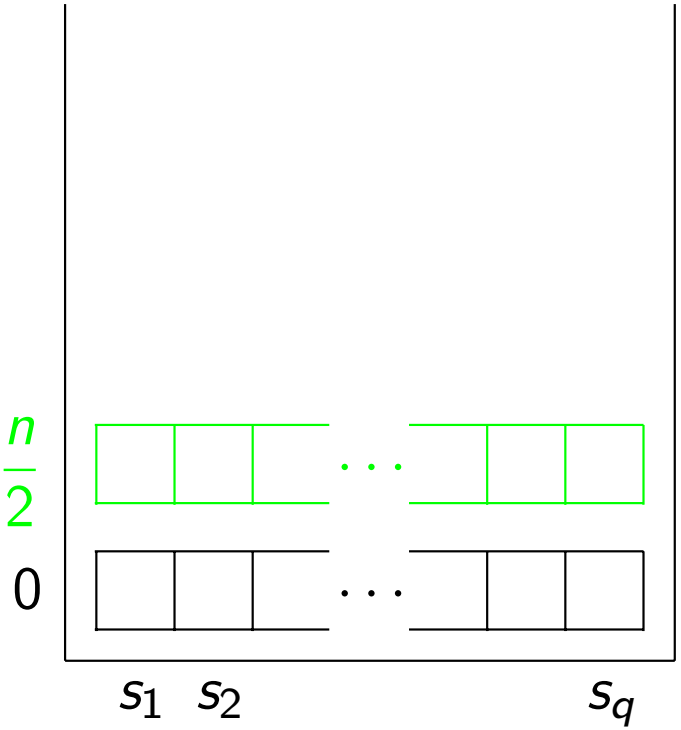
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

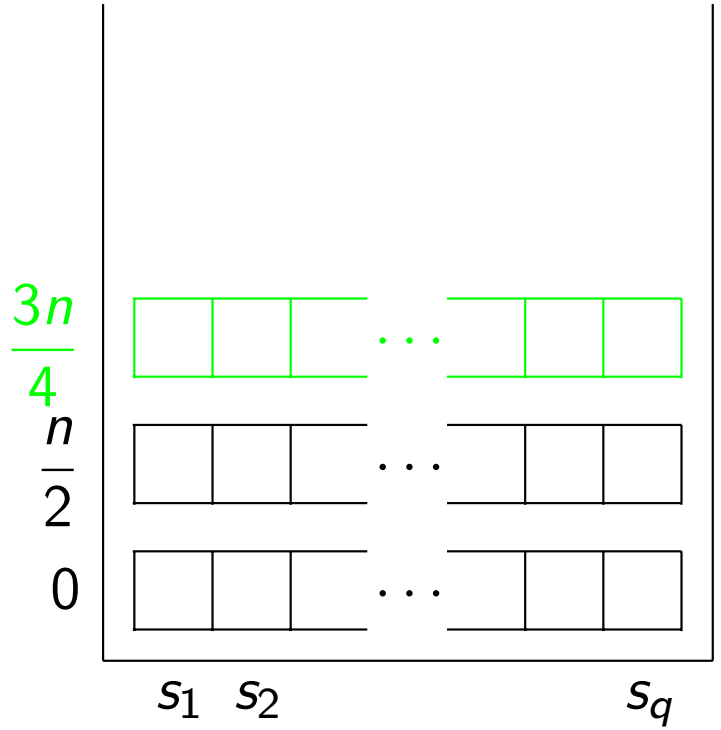
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

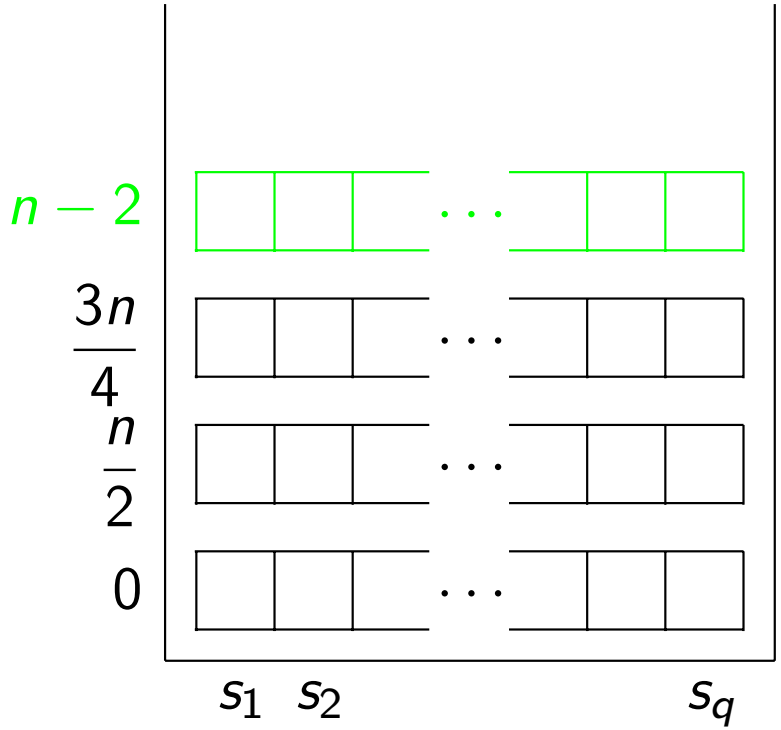
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

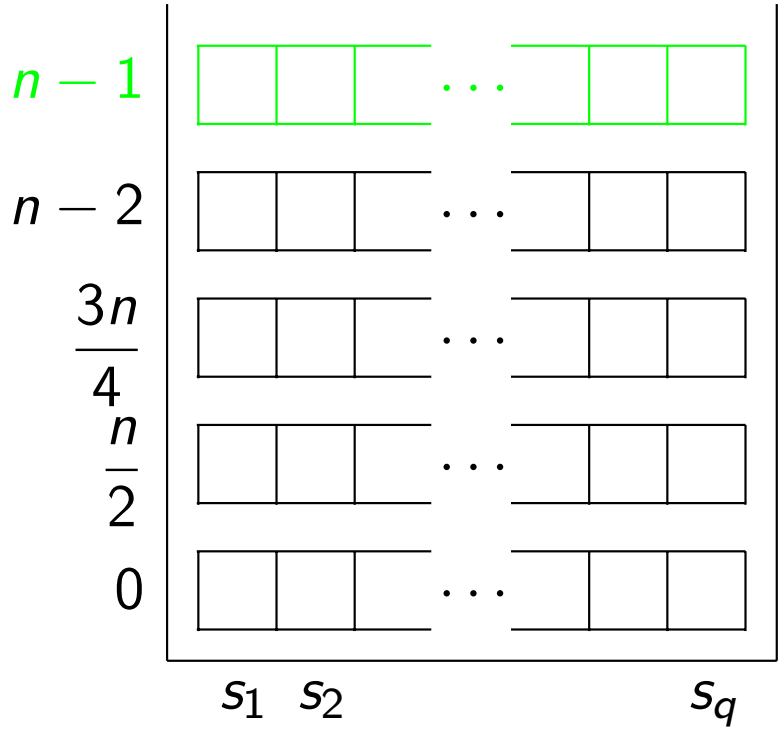
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

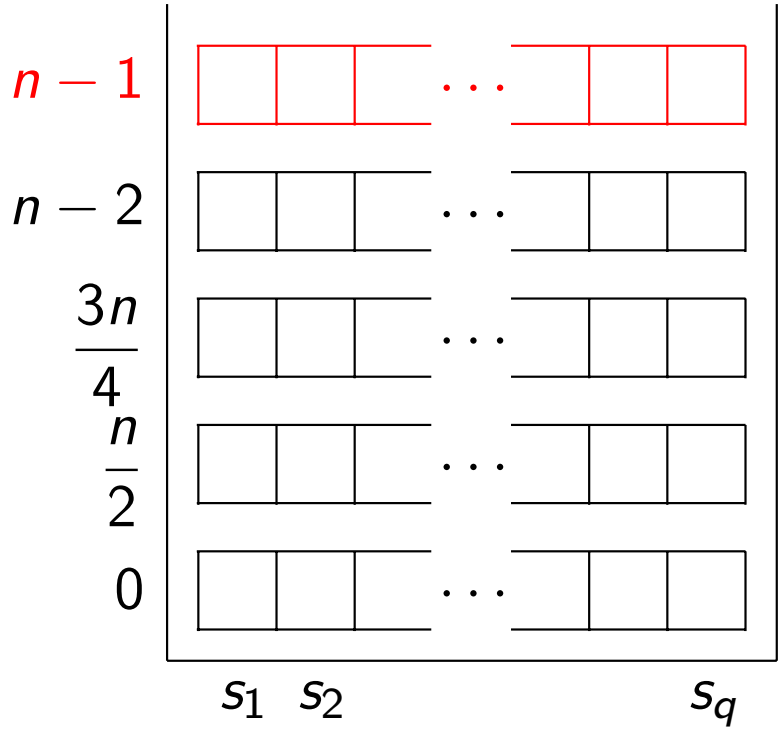
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

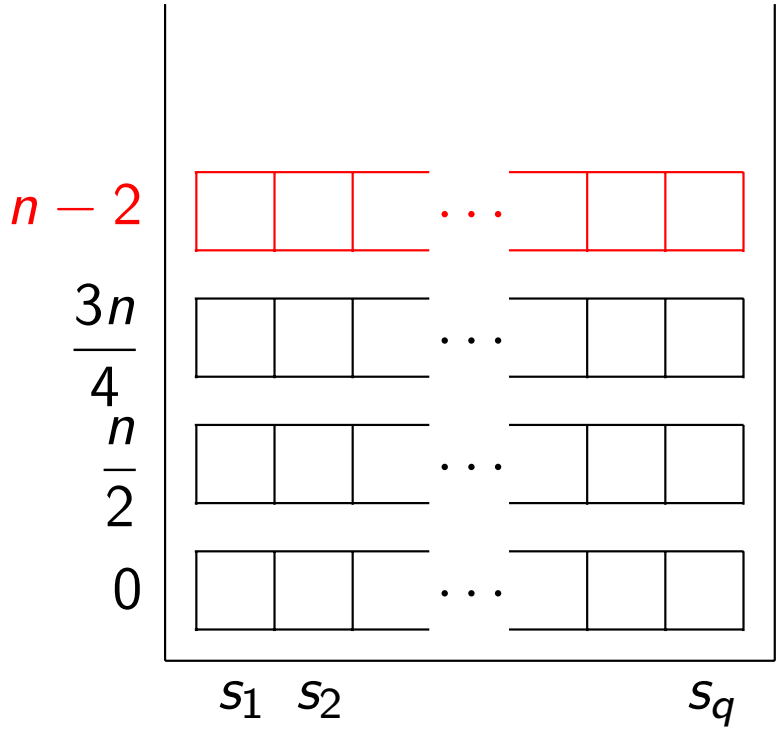
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

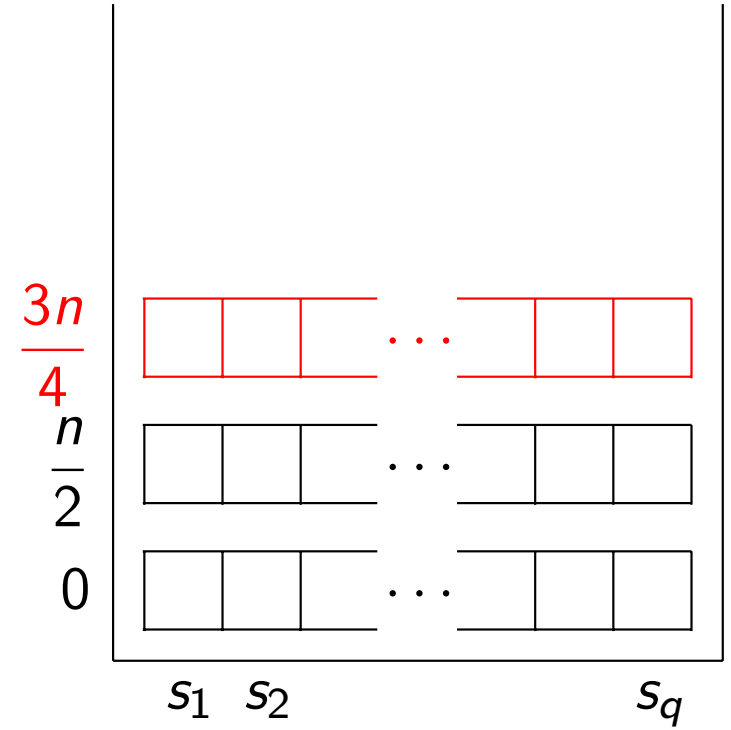
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

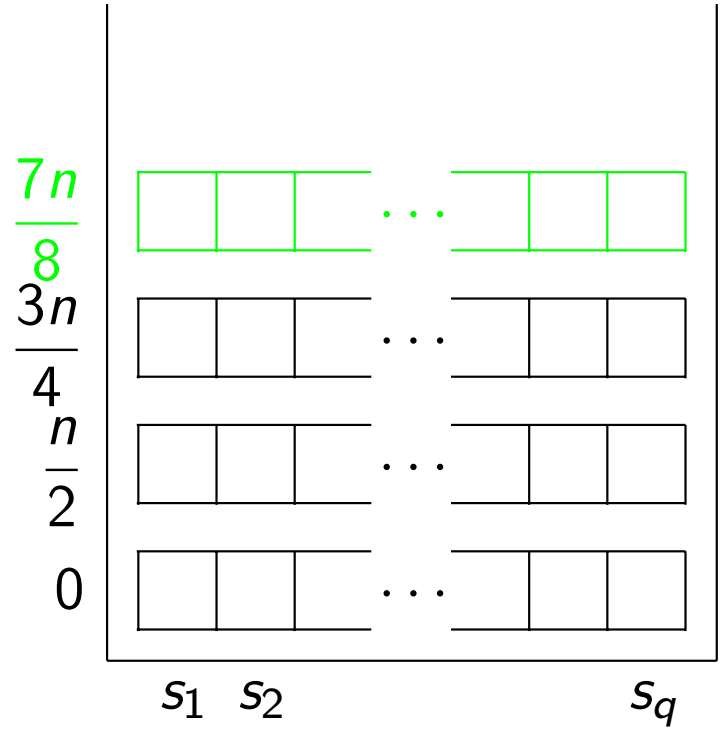
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

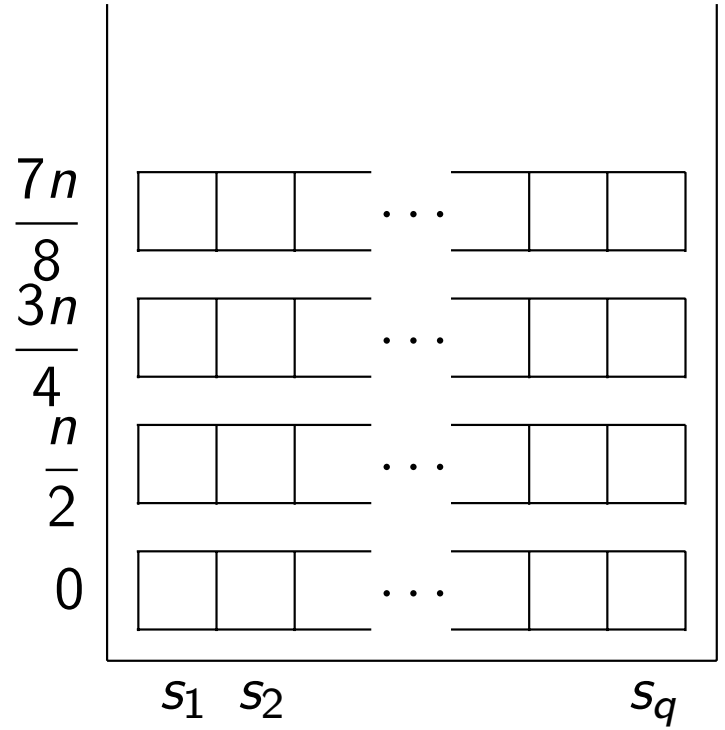
1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.



Algorithme en espace $\mathcal{O}(q \times \log^2 n)$

Principe :

1. Empiler la ligne des $l_s(0)$.
2. Calculer la ligne $n - i$ à partir de la ligne k au sommet de la pile, en empilant les lignes $\frac{n - i + k}{2}$.
3. Choisir le i-ème sommet.

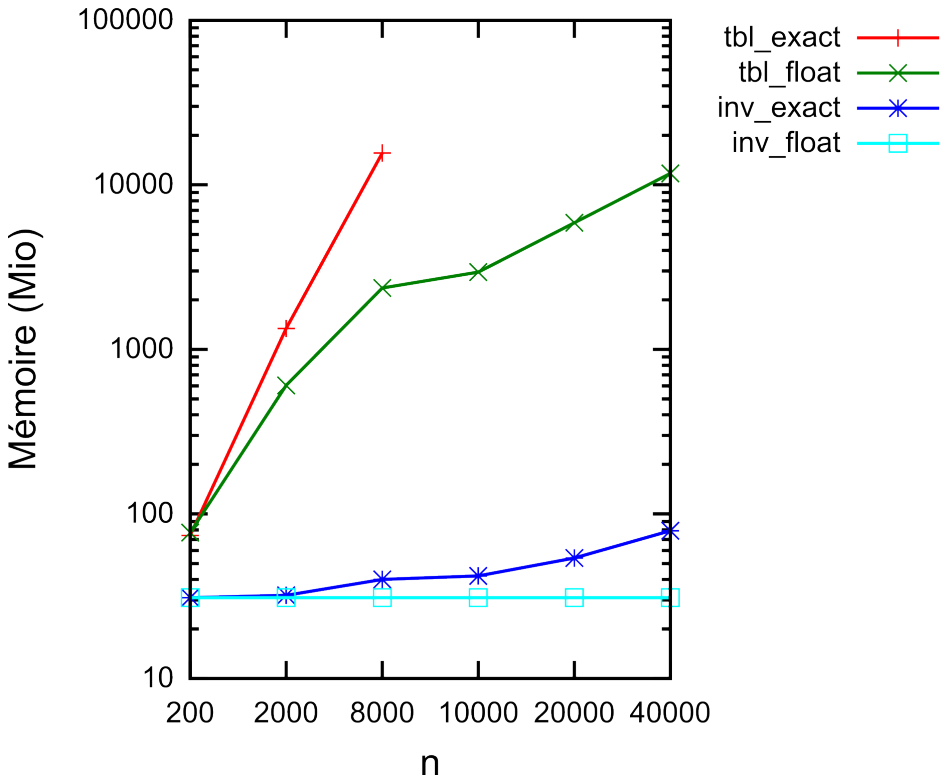


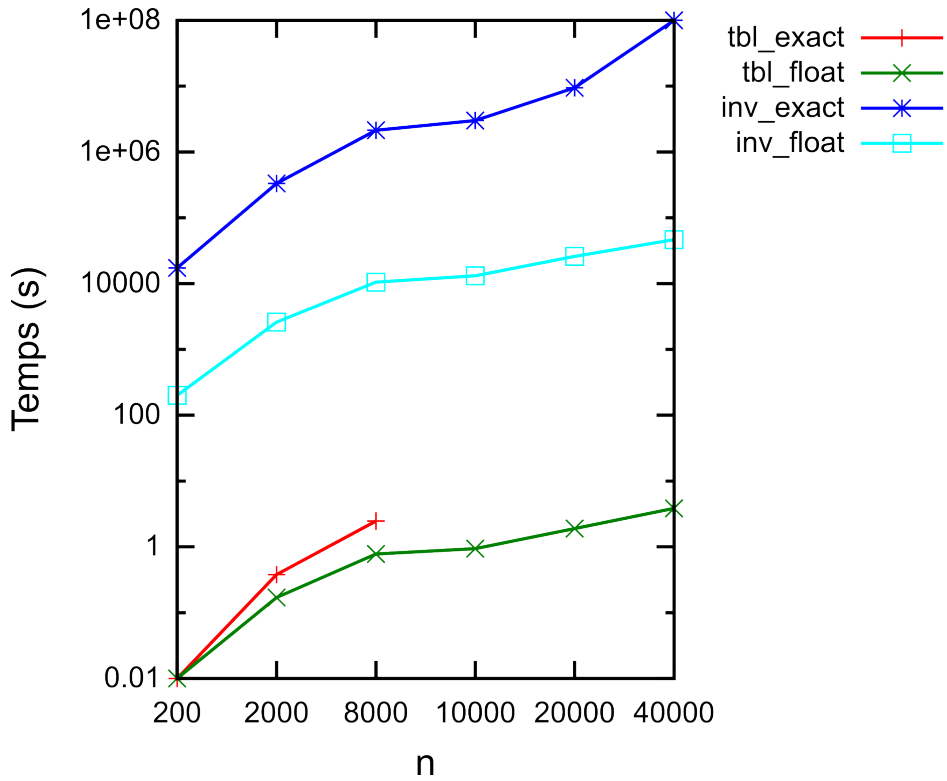
Complexités binaires (avec calcul flottant) :

- temps : $\mathcal{O}(q \times d \times n \log^2 n)$
- espace : $\mathcal{O}(q \times \log^2 n)$

Consommation mémoire et temps d'exécution pour générer 100 chemins

- modèle $\mathrm{vasy\_5\_9}$ de la base VLTS : 5486 états et 9676 transitions.
- légende : avec ou sans table $(\mathrm{tbl/inv})$ _ calcul exact ou flottant $(\mathrm{exact/float})$





Conclusion

Implémentation

Exploration aléatoire garantissant une bonne couverture des chemins du modèle, contrairement à une marche aléatoire isotrope.
Implémentation libre (licence LGPL et dépôt APP) : `rukia.lri.fr`

Résultats :

- En calcul exact, la variante sans table est intéressante car plus économe en mémoire.
- Longueur maximale sur $\mathrm{vasy\_5\_9}$: 8000 $\rightarrow$ 40000.
- En calcul flottant, la variante avec table permet de tirer plus rapidement des chemins plus longs (40000 avec 10Gio de RAM) et avec une très bonne approximation de l'uniformité.
- En calcul flottant, la variante sans table est sujette à l'instabilité numérique et de ce fait non utilisable.

Perspectives

- Comparaison avec l'échantillonneur de Boltzmann : meilleure complexité en espace mais notion d'uniformité différente.
- Tirage uniforme par méthode diviser-pour-régner [Bernardi et Giménez'submitted] : espace en $\mathcal{O}(q^2 \times \log^2 n)$ et temps en $\mathcal{O}(q \times n \log n)$.
- Applications à des fins de simulation, de test, ou de model-checking.

Petites annonces

Aléa la recherche

Jeune doctorant cherche post-doc à partir d'octobre prochain et plus si affinités...

Aléa la bouillabaisse

Bon anniversaire Basile !